Università degli Studi di Bari

Corso di Laurea Triennale in

Informatica e Tecnologie per la Produzione del Software

Aritmetica dei calcolatori e gestione degli errori

Appunti per il Corso di Calcolo Numerico - A.A. 2023/2024

prof. Roberto Garrappa [1]

Indice

[1] Dipartimento di Matematica - Università degli Studi di Bari - E-mail : `roberto.garrappa@uniba.it`.

1 Introduzione al Calcolo Numerico

1.1 Che cos'è il Calcolo Numerico

Il calcolo numerico si occupa di tutte le problematiche legate alla risoluzione di problemi matematici mediante il calcolatore. Sono numerosi i problemi matematici che richiedono l'ausilio del calcolatore per essere risolti. Alcuni problemi tipici che si affrontano in un corso di Calcolo Numerico sono:

- la risoluzione di sistemi di equazioni lineari,
- la ricerca degli zeri di una funzione $f(x)$ (risolvere cioè equazioni non lineari del tipo $f(x) = 0$),
- la ricostruzione di funzioni $f(x)$ a partire dai valori assunti in alcuni punti,
- il calcolo del valore di integrali definiti,

e così via, passando per problemi di maggiore complessità quali il calcolo degli autovalori di una matrice o la risoluzione di equazioni differenziali.

Per poter risolvere questi problemi mediante il calcolatore è necessario individuare i metodi più adatti e studiarne il comportamento (ossia quanto accuratamente risolvono il problema assegnato, con quale costo di calcolo ed occupazione di memoria, ecc.).

Al tempo stesso è necessario conoscere i limiti del calcolatore nel trattare problemi numerici al fine di prevedere come tali limiti influenzano lìacuratezza della soluzione.

Sebbene abbiamo elencato problemi formulati in termini strettamente matematici, è utile sottolineare come la risoluzione di problemi matematici sia richiesta da numerose applicazioni in diversi campi della vita reale (ingegneria, telecomunicazioni, controllo di processi industriali, grafica, elaborazione dei segnali, ecc.).

1.2 Perché studiare il Calcolo Numerico

Il calcolo numerico viene studiato in quanto per risolvere problemi matematici mediante calcolatori è necessario mettere a punto metodologie ad hoc. Le tecniche studiate con l'analisi matematica si rivelano infatti spesso inadeguate per mettere a punto algoritmi e codici da far eseguire al calcolatore.

Il procedimento che porta alla risoluzione di un problema matematico utilizza spesso strumenti quali l'esperienza o l'intuito che è difficile (se non impossibile) codificare in un algoritmo. Per capire meglio questo concetto facciamo alcuni semplici esempi.

Esempio 1.1. Si supponga che si voglia risolvere l'equazione $\sin(2x^2+1) = 0$. Sulla base della nostra esperienza, riconosciamo che $f(x) = \sin(2x^2 + 1)$ è funzione composta $f(x) = h(g(x))$,

dove $h(y) = \sin(y)$ e $g(x) = 2x^2 + 1$. Per cui cercheremo prima le radici della funzione $h(y)$, che sappiamo essere $y = k\pi$, $k = 0, \pm 1, \pm 2, \pm 3, \ldots$, e quindi imporremo che l'argomento di $\sin(y)$ sia pari a questi valori, ossia $2x^2 + 1 = k\pi$, da cui risolveremo $x = \pm\sqrt{\frac{k\pi - 1}{2}}$. L'aver individuato la composizione di funzioni, e l'aver applicato in sequenza due distinte regole per trovare gli zeri di funzioni, è però un'attività difficile da codificare in un codice generale.

Esempio 1.2. Si supponga di voler calcolare il valore dell'integrale definito $\int_0^\pi x\cos(x)dx$. Anche in questo caso, è l'esperienza che ci fa subito notare come la funzione in esame sia il prodotto della funzione $f(x) = x$ per la derivata prima di $g(x) = \sin(x)$ e che, pertanto, risulta immediato applicare la regola di integrazione per parti

$$\int f(x)g'(x)dx = f(x)g(x) - \int f'(x)g(x)dx$$

grazie alla quale risolvere agevolmente

$$\int_0^\pi x\cos(x)dx = -\int_0^\pi \sin(x)dx = \Big[\cos(x)\Big]_0^\pi = -1 - 1 = -2.$$

I due esempi precedenti fanno riferimento a problemi molto semplici, ma evidenziano come possa risultare difficile codificare le tecniche che solitamente utilizziamo per risolvere i problemi matematici. Per questo motivo è necessario sviluppare metodi differenti che sfruttino al meglio le (poche) operazioni che siamo in grado di effettuare in maniera efficiente al calcolatore (ad esempio, il semplice calcolo della funzione in alcuni punti).

Oltre all'aspetto legato alle differenti modalità con cui si devono risolvere i problemi, il calcolo numerico si deve occupare anche di tenere sotto controllo gli errori.

I metodi del calcolo numerico in genere non forniscono soluzioni esatte ma loro approssimazioni. E' necessario pertanto studiare l'accuratezza di ciascun metodo per stimare l'errore che si introduce sulla soluzione.

Ma c'è un'altra fonte di errore, forse più subdola perché meno visibile. Contrariamente a quanto si è soliti pensare, il calcolatore non sempre gestisce in maniera corretta i numeri ed introduce errori anche con le operazioni più semplici; il calcolatore è esso stesso una sorgente di errori per come rappresenta i numeri al suo interno. Dal momento che un algoritmo è una sequenza di operazioni (spesso numerose), la propagazione di errori da una operazione all'altra può portare a risultati disastrosi.

Altri aspetti di cui ci si occupa nello studio del calcolo numerico sono legati alle risorse, in particolare il costo di calcolo (inteso in termini di tempo impiegato dal calcolatore) e l'occupazione di memoria. Si pensi ad un algoritmo che pur essendo in grado di risolvere in maniera accurata un determinato problema, lo faccia con un numero di operazioni tale da richiedere tempi di esecuzione lunghissimi, o richiedendo una occupazione di memoria superiore a quella disponibile. E' vero che le risorse che si hanno oggi a disposizione (in termini di velocità di elaborazione e quantità di memoria) sono molto più ampie che in passato, ma è anche vero che si cerca di risolvere problemi sempre più grossi e sempre più complessi per cui non si può trascurare di tenere sotto controllo l'efficienza computazionale degli algoritmi.

Possiamo pertanto riassumere i principali aspetti di cui si occupa il calcolo numerico:

- messa a punto di metodi ad hoc per la risoluzione di problemi matematici mediante calcolatore ;
- studio dell'accuratezza dei risultati, degli errori introdotti e della loro propagazione;

- analisi del costo di calcolo in termini di tempi di esecuzione;
- analisi della memoria richiesta dagli algoritmi.

2 Gli errori nel calcolo numerico

Uno dei problemi più grossi nello sviluppo di software numerico è quello legato agli errori ed alla necessità di tenerli sotto controllo. Possiamo, in linea generale, individuare due classi di errori che si introducono (e quindi si propagano) quando si sviluppa software per la risoluzione di problemi numerici

1. *errori analitici,*
2. *errori di arrotondamento.*

2.1 Errori analitici

Gli *errori analitici* (detti anche *errori di troncamento*) si generano quando il metodo utilizzato risolve esattamente il problema solo a seguito di un procedimento limite.

Ad esempio, i metodi iterativi utilizzati per il calcolo della soluzione di sistemi lineari, per la risoluzione di equazioni non lineari o, ancora, per il calcolo degli autovalori di una matrice, generano una successione di approssimazioni $\{x_k\}_k = \{x_0, x_1, x_2, \dots\}$ che solo per $k \to \infty$ converge alla soluzione $x^\star$ del problema. Dal momento che non è possibile implementare al calcolatore un procedimento infinito, si dovrà terminare l'algoritmo dopo un numero finito K di iterazioni, una volta calcolata una approssimazione x_K ritenuta sufficientemente vicina alla soluzione del problema. Dal momento che quella calcolata dopo K passi è solo una approssimazione della soluzione del problema, l'assumere x_K come soluzione invece di quella teorica $x^\star$ introduce un errore che sarà tanto più grande quanto piú x_K è lontano da $x^\star$.

Esempi analoghi si possono proporre per il calcolo di integrali o per le formule relative all'interpolazione ed all'approssimazione.

Lo studio di questi errori dipende essenzialmente dal metodo utilizzato, e poco si può dire in linea generale, se non entrando nel merito di ciascun metodo, come faremo quando studieremo il singolo metodo.

In generale anche il calcolo, mediante elaboratore, delle normali funzioni elementari è soggetto ad errori di questo tipo. Infatti, una delle tecniche alla base del calcolo di funzioni elementari, è basata sullo sviluppo in serie di Taylor, per cui semplici funzioni elementari vengono calcolate mediante sviluppi in serie come, ad esempio

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!} \qquad \sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}.$$

Poiché non è pensabile effettuare un numero infinito di operazioni, queste sommatorie dovranno essere prima o poi troncate. Per cui il valore che sarà effettivamente calcolato non sarà quello di $\exp(x)$ o $\sin(x)$ ma una loro approssimazione data da

$$\exp_K(x) = \sum_{k=0}^{K} \frac{x^k}{k!} \qquad \sin_K(x) = \sum_{k=0}^{K} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

e, per quanto K possa essere grande, avremo comunque introdotto un errore. Nella tabella che segue abbiamo riportato i risultati dell'approssimazione della funzione $\exp(x)$ nel punto $x = 1.0$ per diversi valori di K e mostrato l'errore commesso.

K	$\exp_K(x)$	$\lvert \exp(x) - \exp_K(x) \rvert$
1	2.0	7.18×10^{-1}
2	2.5	2.18×10^{-1}
3	2.6666666666666665	5.16×10^{-2}
4	2.7083333333333330	9.94×10^{-3}
5	2.7166666666666663	1.61×10^{-3}
10	2.7182818011463845	2.73×10^{-8}
15	2.7182818284589949	5.06×10^{-14}

2.2 Errori di arrotondamento

Gli errori di arrotondamento sono dovuti al fatto che il calcolatore, contrariamente a quanto si è soliti pensare, non esegue i calcoli in modo preciso ma approssimato. Così come molto spesso approssimato è il modo in cui i numeri vengono immagazzinati nella memoria del calcolatore.

Infatti, un numero reale ha spesso una rappresentazione estremamente lunga se non proprio infinita, e quando deve essere adoperato da un calcolatore è possibile utilizzare solamente un numero contenuto, e comunque finito, di cifre. Pertanto non sempre un numero reale potrà essere rappresentato e memorizzato al calcolatore ed il più delle volte si dovrà utilizzare una sua approssimazione. Si pensi al numero π

$$\pi = 3,14159265358979323846264338327950288419716939937510582097494459 2\ldots$$

che come è noto ha un numero infinito di cifre decimali. Non potendo certamente memorizzarle tutte, solo una parte di questo numero potrà essere memorizzata e pertanto invece che il π si memorizzerà una sua approssimazione, introducendo così un errore.

Inoltre, dal momento che il calcolatore utilizza una aritmetica di tipo binario, e quindi in base 2, spesso anche numeri reali che hanno una rappresentazione decimale finita, una volta convertiti in binario possono avere una rappresentazione infinita che come tale non potrà essere contenuta in memoria, e pertanto il numero potrà essere rappresentato solamente in modo approssimato. Ad esempio, il numero decimale 0.1 quando convertito in rappresentazione binaria assume la seguente forma con un numero infinito di cifre

$$0,000110011001100110011001100110011001100110011001100\ldots;$$

tale numero per poter essere memorizzato nella memoria del calcolatore che ha un numero finito di locazioni, dovrà necessariamente essere sostituito da un altro numero (vicino ma comunque diverso) che abbia un numero finito di cifre. Questa sostituzione, come si intuisce facilmente, introduce un errore.

2.3 Altre forme di errore

Oltre a quelli citati, che costituiscono oggetto di studio approfondito nel nostro corso, vi sono altre forme di errore che entrano in gioco nel calcolo numerico, e che è opportuno tenere in considerazione.

Tra questi segnaliamo gli **errori sui dati**, legati al fatto che i dati rinvengono spesso da misurazioni e pertanto, come tali, sono affetti da errore. Possiamo far rientrare questi errori tra gli errori inerenti (che introdurremo più avanti) e quindi studiare il condizionamento del problema rispetto ad errori di misurazione.

Un ulteriore forma di errore è quella introdotta con le **semplificazioni nel modello rappresentato**. Generalmente quando si vuole rappresentare un sistema dinamico, ad esempio attraverso delle equazioni differenziali, si è soliti fare delle semplificazioni per rendere più agevole la costruzione del modello stesso. Ad esempio, nel caso della descrizione del moto di un corpo, molto spesso si suppone che non vi sia attrito e quindi si sceglie un modello semplificato. Naturalmente queste semplificazioni hanno una grossa influenza sui risultati del calcolo anche se, non dipendendo dai metodi numerici, dal software o dall'hardware utilizzato, non sono oggetto di studio specifico nel calcolo numerico.

2.4 Misurazione dell'errore : errore assoluto ed errore relativo

A causa della presenza di errori, invece che il valore esatto di un dato x dobbiamo attenderci di avere una sua approssimazione $\tilde{x}$. E' naturale quindi porsi il problema di confrontare i due valori per dare una misurazione della bontà dell'approssimazione e dell'errore che abbiamo nell'utilizzare $\tilde{x}$ al posto di x.

Vi sono diversi modi per misurare l'errore. Una prima misurazione è costituita da quello che chiamiamo **errore assoluto**

$$e_A(x) = |x - \tilde{x}|$$

nel caso in cui x e $\tilde{x}$ sono scalari, oppure

$$e_A(x) = \|x - \tilde{x}\| \; ,$$

nel caso in cui x e $\tilde{x}$ sono vettori o matrici e dove $\| \cdot \|$ è una qualche norma (studieremo in seguito il concetto di norma, per il momento ci basta sapere che una norma permette di generalizzare il valore assoluto ad una struttura di dati quali i vettori o le matrici).

L'errore assoluto però non fornisce sempre una misura adeguata della bontà dell'approssimazione in quanto dipende dall'ordine di grandezza del dato stesso: sarà cioè più elevato se x è grande e più contenuto se x è piccolo.

Esempio 2.1. Se abbiamo $x = 1'000'000$ ed una sua approssimazione $\tilde{x} = 1'000'000{,}5$ l'errore assoluto è pari a $|x - \tilde{x}| = 0{,}5$, mentre se abbiamo $y = 2$ ed una sua approssimazione $\tilde{y} = 2{,}1$ l'errore assoluto è pari a $|y - \tilde{y}| = 0.1$. Si verrebbe indotti a ritenere che $\tilde{y}$ approssimi y meglio di quanto non faccia invece $\tilde{x}$ con x. La nostra esperienza quotidiana ci dice invece che vale il contrario: si pensi alla differenza che passa tra aggiungere (o togliere) 50 centesimi al prezzo di un bene che costa mille euro e tra l'aggiungere (o togliere) 10 centesimi al prezzo di un bene che costa invece solo 2 euro.

Laddove è possibile, è preferibile ricorrere all'**errore relativo** che misura l'errore in rapporto alle grandezze in esame. Se $x \neq 0$ (oppure $\|x\| \neq 0$ nel caso vettoriale) l'errore relativo si definisce come

$$e_R(x) = \frac{|x - \tilde{x}|}{|x|}, \quad e_R(x) = \frac{\|x - \tilde{x}\|}{\|x\|}.$$

Con riferimento all'esempio precedente, i due errori relativi sono dati da

$$
\begin{aligned}
e_R(x) &= \frac{|x-\tilde{x}|}{|x|} = \frac{|1\text{'}000\text{'}000 - 1\text{'}000\text{'}000{,}5|}{|1\text{'}000\text{'}000|} = 0{,}0000005 \\
e_R(y) &= \frac{|y-\tilde{y}|}{|y|} = \frac{|2-2{,}1|}{|2|} \approx 0{,}05\,,
\end{aligned}
$$

da cui possiamo dedurre come l'approssimazione $\tilde{x}$ per x sia più accurata rispetto a quella fornita da $\tilde{y}$ per y.

L'errore relativo può essere calcolato solamente quando $|x| \neq 0$ (oppure $\|x\| \neq 0$ nel caso vettoriale). Per evitare di dover effettuare tale verifica possiamo utilizzare l'**errore misto**

$$
e(x) = \frac{|x-\tilde{x}|}{1+|x|}, \quad e(x) = \frac{\|x-\tilde{x}\|}{1+\|x\|}
$$

che approssima l'errore assoluto se $|x|$ (o $\|x\|$) è piccolo e l'errore relativo in caso contrario.

2.5 Propagazione degli errori: condizionamento dei problemi

Abbiamo osservato come i dati con cui di solito si lavora al calcolatore siano affetti da errori di diversa natura. E' importante pertanto studiare la sensibilità dei diversi problemi agli errori sui dati: è naturale infatti aspettarsi che, a fronte di dati affetti da errori, anche i risultati siano in qualche modo affetti da errori ed è necessario accertarsi che gli errori sui risultati non crescano troppo ma si mantengano dello stesso ordine di grandezza degli errori sui dati, altrimenti il problema non potrà essere risolto. Introduciamo pertanto la seguente definizione.

Definizione 2.1. *Un problema si dice* **ben condizionato** *se a piccoli errori sui dati corrispondono piccoli errori sui risultati, altrimenti si dirà mal condizionato.*

Il buon o mal condizionamento è una caratteristica intrinseca del problema e non dipende dal metodo utilizzato per risolverlo. E' evidente che un problema mal condizionato sarà risolto in modo poco accurato qualunque metodo si utilizzi. In tal caso sarà necessario sostituire al problema uno equivalente (cioè che abbia la stessa soluzione) ma che risulti meglio condizionato.

Vediamo come si può studiare il condizionamento di alcuni semplici problemi. Per fare ciò da ora in poi l'approssimazione $\tilde{x}$ del dato x sarà indicata come l'effetto della perturbazione del dato x mediante una quantità (generalmente piccola) δ_x che potrà essere positiva o negativa, ossia $\tilde{x} = x + \delta_x$.

Somma di due numeri

Supponiamo di dover effettuare la somma $s = x + y$ di due numeri x e y, di cui non si dispone il valore esatto ma solamente di un valore affetto da perturbazione $\tilde{x} = x + \delta_x$ e $\tilde{y} = y + \delta_y$, con un errore relativo sui dati che è pari a

$$
e_R(x) = \frac{|x-\tilde{x}|}{|x|} = \frac{|x-(x+\delta_x)|}{|x|} = \frac{|\delta_x|}{|x|} \;\text{ e }\; e_R(y) = \frac{|y-\tilde{y}|}{|y|} = \frac{|y-(y+\delta_y)|}{|y|} = \frac{|\delta_y|}{|y|}.
$$

Operando su dati affetti da errori, anche il risultato dell'operazione di somma risentirà degli effetti degli errori, per cui non otterremo come risultato il valore teorico $s = x + y$ ma il valore

dato da $\tilde{s} = \tilde{x} + \tilde{y} = (x + \delta_x) + (y + \delta_y)$. Pertanto possiamo andare a calcolare l'errore relativo sul risultato come

$$\begin{aligned} e_R(s) &= \frac{|s - \tilde{s}|}{|s|} = \frac{|(x + y) - ((x + \delta_x) + (y + \delta_y))|}{|x + y|} = \\ &= \frac{|(x + y) - (x + y) - \delta_x - \delta_y)|}{|x + y|} = \frac{|\delta_x + \delta_y|}{|x + y|} \leq \frac{|\delta_x|}{|x + y|} + \frac{|\delta_y|}{|x + y|}. \end{aligned}$$

Affinchè si abbia una relazione tra l'errore relativo sul risultato e gli errori sui dati, moltiplichiamo e dividiamo il primo termine per $|x|$ ed il secondo termine per $|y|$, ed otteniamo

$$e_R(s) \leq \frac{|\delta_x|}{|x|} \cdot \frac{|x|}{|x + y|} + \frac{|\delta_y|}{|y|} \cdot \frac{|y|}{|x + y|} = \frac{|x|}{|x + y|}\, e_R(x) + \frac{|y|}{|x + y|}\, e_R(y).$$

Gli errori relativi $e_R(x)$ e $e_R(y)$ sui dati x e y vengono pertanto amplificati rispettivamente dai fattori

$$\frac{|x|}{|x + y|} \quad \text{e} \quad \frac{|y|}{|x + y|}$$

e possiamo affermare che, se $x + y$ è grande, allora i coefficienti di amplificazione sono piccoli e quindi gli errori sui dati non crescono: in questo caso il problema è ben condizionato; se invece $x + y$ è piccolo abbiamo dei coefficienti di amplificazione elevati e pertanto il risultato ne risulterà perturbato: in questo caso il problema è mal condizionato.

La semplice operazione di somma tra due numeri risulta essere mal condizionata laddove si vanno a sommare valori tra loro vicini ma di segno opposto, e quindi quando viene effettuata la differenza tra valori vicini che risulta pertanto essere una operazione mal condizionata.

Esempio 2.2. *Si considerino i due numeri* 2,34562 *e* 2,34563 *e che siano entrambi affetti da un errore relativo approssimativamente pari a* 1.0×10^{-14}. *Si stimi di quanto viene amplificato l'errore quando se ne calcola la loro differenza.*

I fattori di amplificazione della differenza, espressa come somma di $x = 2{,}34562$ e $-y = -2{,}34563$, sono pari a

$$\frac{|x|}{|x + (-y)|} = \frac{|x|}{|x - y|} = \frac{2{,}34562}{0{,}00001} \approx 2{,}3 \times 10^5, \qquad \frac{|y|}{|x + (-y)|} = \frac{2{,}34563}{0{,}00001} \approx 2{,}3 \times 10^5$$

e pertanto per l'errore relativo della differenza abbiamo

$$e_R(s) \leq 2{,}3 \times 10^5 \times 10^{-14} + 2{,}3 \times 10^5 \times 10^{-14} = 2{,}3 \times 10^{-9} + 2{,}3 \times 10^{-9} = 4{,}6 \times 10^{-9}$$

e quindi l'errore sui dati avente un ordine di grandezza pari a 10^{-14} è stato amplificato producendo un errore sui risultati con un ordine di grandezza pari a 10^{-9}. Si tratta chiaramente di un problema mal condizionato.

Prodotto di due numeri

Consideriamo adesso il prodotto $p = x \cdot y$ di due numeri x e y ed il prodotto $\tilde{p} = \tilde{x} \cdot \tilde{y}$ ottenuto moltiplicando i corrispondenti valori perturbati $\tilde{x} = x + \delta_x$ e $\tilde{y} = y + \delta_y$. L'errore relativo sul risultato è dato da

$$
\begin{aligned}
e_R(p) &= \frac{|p-\tilde{p}|}{|p|} = \frac{|(x \cdot y) - (x+\delta_x)\cdot(y+\delta_y)|}{|x\cdot y|} \\
&= \frac{|(x\cdot y) - (x \cdot y) - x \cdot \delta_y - y \cdot \delta_x - \delta_x \cdot \delta_y)|}{|x \cdot y|} \leq \\
&\leq \frac{|x\delta_y|}{|x\cdot y|} + \frac{|y\delta_x|}{|x\cdot y|} + \frac{|\delta_x\delta_y|}{|x\cdot y|} = \frac{|\delta_y|}{|y|} + \frac{|\delta_x|}{|x|} + \frac{|\delta_x|}{|x|} \cdot \frac{|\delta_y|}{|y|} = \\
&= e_R(x) + e_R(y) + e_R(x) \cdot e_R(y).
\end{aligned}
$$

Se assumiamo che $e_R(x)$ e $e_R(y)$ siano quantità piccole (come appare certamente ragionevole), allora il prodotto $e_R(x) \cdot e_R(y)$ può essere trascurato rispetto alla somma di $e_R(x)$ e $e_R(y)$ e pertanto abbiamo

$$e_R(p) \lessapprox e_R(x) + e_R(y)$$

che ci indica che gli errori relativi sui dati non subiscono alcuna significativa amplificazione (abbiamo utilizzato il simbolo $\lessapprox$ per indicare che vale la maggiorazione in forma approssimata, ossia solo quando $e_R(x) \cdot e_R(y)$ è piccolo da poter essere trascurato). L'errore relativo sul risultato è pertanto dello stesso ordine di grandezza dell'errore sui dati ed il problema del calcolo del prodotto è quindi ben condizionato.

Calcolo del valore di una funzione $f(x)$

Consideriamo una funzione $f(x)$ che supponiamo sufficientemente regolare (continua e dotata di derivate) e studiamo come si gli errori propagano sul dato in ingresso nel calcolo della funzione. Grazie al teorema di Taylor, sappiamo che

$$f(x+\delta_x) = f(x) + \delta_x f'(x) + R_2(x)\mathcal{O}(\delta_x^2)$$

con il resto $R_2(x)$ che tende a zero, per $\delta_x \to 0$, più velocemente rispetto a δ_x; pertanto, se assumiamo δ_x piccolo, possiamo trascurare il resto $R_2(x)$ che risulta essere piccolo rispetto alle altre quantità, e quindi approssimare $f(x+\delta_x) \approx f(x) + \delta_x f'(x)$ da cui otteniamo l'approssimazione dell'errore

$$e_R(f(x)) \approx \left| \frac{f(x) - f(x+\delta_x)}{f(x)} \right| = \left| \frac{f(x) - f(x) - \delta_x f'(x)}{f(x)} \right| = |\delta_x| \left| \frac{f'(x)}{f(x)} \right|$$

Pertanto, moltiplicando e dividendo per $|x|$, abbiamo

$$e_R(f(x)) \approx \left| \frac{\delta_x}{x} \right| \left| x \frac{f'(x)}{f(x)} \right| = e_R(x) \cdot \left| x \frac{f'(x)}{f(x)} \right|$$

e deduciamo che l'errore relativo sui dati viene amplificato di un fattore $\left| x \frac{f'(x)}{f(x)} \right|$ che rappresenta l'indice di condizionamento del problema.

Il calcolo del valore di una funzione in un punto x sarà pertanto ben condizionato quando la quantità $\left| x \frac{f'(x)}{f(x)} \right|$ risulterà piccola, altrimenti sarà mal condizionato.

Non si può dire a priori se il calcolo di $f(x)$ sia ben condizionato o meno in quanto ciò dipende non solo dalla funzione f ma, in molti casi, anche dal punto x in cui il calcolo viene richiesto.

Esempi: indice di condizionamento per alcune funzioni. Se consideriamo la funzione $f(x) = \sqrt{x}$, dal momento che $f'(x) = \frac{1}{2\sqrt{x}}$ abbiamo un indice di condizione che è dato da

$$\left| x\frac{f'(x)}{f(x)} \right| = \left| x\frac{1}{2\sqrt{x}}\frac{1}{\sqrt{x}} \right| = \left| x\frac{1}{2x} \right| = \frac{1}{2}$$

e pertanto il problema è sempre ben condizionato. Se invece consideriamo la funzione $f(x) = \sin(x)$, osserviamo che la derivata prima è data da $f'(x) = \cos(x)$ e quindi l'indice di condizione può essere calcolato come

$$\left| x\frac{f'(x)}{f(x)} \right| = \left| x\frac{\cos(x)}{\sin(x)} \right| = \left| \frac{x}{\tan(x)} \right|$$

che dipende dal valore di x. Per la funzione $f(x) = e^x$ abbiamo infine che $f'(x) = e^x$ e pertanto, essendo

$$\left| x\frac{f'(x)}{f(x)} \right| = \left| x\frac{e^x}{e^x} \right| = |x| \ ,$$

il suo calcolo sarà ben condizionato per valori piccoli di x e mal condizionato per valori grandi di x.

Esercizio 2.1. Dati i valori $x = 32{,}75642$ e $y = 32{,}75641$ si fornisca una stima di quanto verranno amplificati eventuali errori sui dati calcolando la differenza $x - y$. Effettuare il calcolo nel caso in cui $e_R(x) = 1{,}0 \times 10^{-14}$ e $e_R(y) = 2{,}0 \times 10^{-14}$.

Esercizio 2.2. Si dica se il calcolo della funzione x^3 è ben condizionato o meno e se ne spieghi il motivo.

Esercizio 2.3. Si studi il condizionamento del calcolo della funzione $\cos x$ e si stabilisca in corrispondenza di quali valori della x il problema risulta mal condizionato.

Esercizio 2.4. Si studi il condizionamento del calcolo delle seguenti funzioni e si stabilisca in corrispondenza di quali valori della x il problema risulti eventualmente mal condizionato

- $f(x) = x(x^2 - 1)$
- $f(x) = x(x^2 + 1)$
- $f(x) = \frac{1}{2-3x}$

3 La rappresentazione dei numeri al calcolatore

Una fonte di introduzione di errori nell'esecuzione di programmi di tipo numerico è legata al fatto che i calcolatori utilizzano un modo di rappresentare i dati differente da quello a cui siamo abituati. Questa differenza si evidenzia essenzialmente in due aspetti:

- uso di una base differente;
- utilizzo di un numero finito di cifre.

Nella nostra società si è affermato ormai da alcuni secoli il sistema di numerazione **decimale** basato sulle dieci cifre

$$0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9,$$

e di natura **posizionale** in quanto il valore da attribuire ad ogni cifra è dato dalla sua posizione nel numero. Ad esempio, il numero

$$\begin{array}{ccccc} 4 & 3 & 7 & 1 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 4\cdot 10^4 & 3\cdot 10^3 & 7\cdot 10^2 & 1\cdot 10^1 & 6\cdot 10^0 \end{array}$$

equivale a $4\cdot 10^4 + 3\cdot 10^3 + 7\cdot 10^2 + 1\cdot 10^1 + 6\cdot 10^0 = 43716$.

I calcolatori non sono però in grado di utilizzare direttamente questa rappresentazione numerica, in quanto al loro interno utilizzano una **rappresentazione binaria**, cioè in base 2, basate sulle sole cifre

$$0\ 1$$

e sempre di natura posizionale; pertanto il numero binario $11011,101$

$$\begin{array}{cccccccc} 1 & 1 & 0 & 1 & 1 & , \ 1 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1\cdot 2^4 & 1\cdot 2^3 & 0\cdot 2^2 & 1\cdot 2^1 & 1\cdot 2^0 & 1\cdot 2^{-1} & 0\cdot 2^{-2} & 1\cdot 2^{-3} \end{array}$$

equivale a $1\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0+1\cdot 2^{-1}+1\cdot 2^{-3} = 16+8+2+1+0,5+0,125 = 27,625.$

Naturalmente per ogni valore che immettiamo in notazione decimale, il calcolatore dovrà effettuare la conversione al suo interno in notazione binaria, eseguire le eventuali operazioni, tutte mediante l'aritmetica binaria, e quindi effettuare nuovamente la conversione in formato decimale per rendere il risultato comprensibile all'utente.

Un'aspetto da non trascurare nella fase di conversione è quello legato al fatto che un calcolatore deve necessariamente rappresentare i numeri attraverso un numero finito e fissato a priori di cifre, e questo può portare a diversi tipi di problemi.

3.1 La rappresentazione normalizzata

Un numero decimale è costituito da una parte intera (le cifre che precedono la virgola) e da una parte decimale (le cifre che seguono la virgola). Esempi di numeri decimali sono

$$x_1 = 12,3456$$
$$x_2 = 0,345232$$
$$x_3 = 423,23423$$

Molto spesso, per comodità, uno stesso numero, specie se molto grande o molto piccolo e con numerosi zeri, quali ad esempio numeri del tipo

$$x = 3400000000, \quad y = 0,000000000056$$

vengono rappresentati mediante un prodotto in cui gli zeri diventano l'esponente di una potenza della base (nel nostro esempio la base è 10), come

$$x = 3,4\times 10^9, \quad y = 5,6\times 10^{-11}.$$

E' evidente come questo tipo di rappresentazione permetta di scrivere i numeri in una forma più compatta. Osserviamo però che questo tipo di rappresentazione non è unica. In generale, moltiplicando e dividendo un numero decimale per la base della sua rappresentazione elevata ad un qualunque altro esponente, si ottiene uno spostamento della virgola a sinistra o a destra, come negli esempi che seguono

$$\begin{aligned}
x_1 &= 12,3456 = 12,3456 \times \frac{10}{10} = 1,23456 \times 10 = \\
&= 12,3456 \times \frac{10^2}{10^2} = 0,123456 \times 10^2 = \dots \\
x_2 &= 0,345232 = 0,345232 \times \frac{10^2}{10^2} = 34,5232 \times 10^{-2} \\
&= 0,345232 = 0,345232 \times \frac{10^3}{10^3} = 0,000345232 \times 10^3 = \dots \\
x_3 &= 423,23423 = 423,23423 \times \frac{10^3}{10^3} = 0,42323423 \times 10^3 \\
&= 423,23423 \times \frac{10^2}{10^2} = 42323,423 \times 10^{-2} = \dots
\end{aligned}$$

Pertanto uno stesso numero può essere rappresentato in diversi modi tra loro equivalenti, nel senso che indicano tutti la stessa quantità.

Tra tutte le possibili rappresentazioni che abbiamo a disposizione, una particolarmente utilizzata in matematica e nei calcolatori è la **rappresentazione normalizzata**, per ottenere la quale si moltiplica e divide il numero per la base elevata ad un opportuno esponente in modo tale che:

- la parte intera del numero sia nulla,
- la prima cifra dopo la virgola sia diversa da zero.

La rappresentazione normalizzata dei numeri dell'esempio precedente è pertanto data da

$$\begin{aligned}
x_1 &= 0,123456 \times 10^2 \\
x_2 &= 0,345232 \times 10^0 \\
x_3 &= 0,42323423 \times 10^3 \ .
\end{aligned}$$

In questo modo si conseguono due obiettivi fondamentali:

- numeri di diversa grandezza hanno una rappresentazione tra loro omogenea, che occupa la stessa quantità di memoria (si pensi a quanto ciò possa essere vantaggioso dovendo manipolare i numeri al calcolatore e quindi in modo automatico);
- si evita di sprecare memoria per rappresentare cifre non significative (gli zeri) come nel caso di numeri molto piccoli (è il caso, ad esempio, di un numero del tipo $0,0000000043535$ che viene più convenientemente rappresentato come $0,43535 \times 10^{-8}$) o di numeri molto grandi.

Il fatto che la parte esponente di un numero nella rappresentazione normalizzata sia una potenza di 10 non è casuale ma dipende dal fatto che la base della numerazione è proprio 10. Se utilizzassimo una numerazione in base 2 (come nel caso dei calcolatori) la parte esponente avrebbe potenze di 2.

In generale se β è la base della rappresentazione (nell'uso corrente è di solito $\beta = 10$, mentre nell'aritmetica dei calcolatori, che è di tipo binario, abbiamo $\beta = 2$), allora un numero in formato normalizzato si presenta come

$$x = \pm\, 0, d_1 d_2 d_3 \ldots \times \beta^E \;, \; d_1 \neq 0 \tag{1}$$

dove $d_1, d_2, d_3, \ldots$ sono dette cifre della **mantissa**, E prende il nome di **caratteristica** e la quantità β^E viene chiamata **parte esponente**. Quindi un numero normalizzato è caratterizzato univocamente da quattro elementi

$$\text{numero normalizzato} \; : \; \begin{cases} \text{base} \\ \text{segno} \\ \text{cifre della mantissa} \\ \text{caratteristica} \end{cases}$$

Le cifre d_i della mantissa dipendono dalla base e variano nell'insieme $\{0, 1, \ldots, \beta-1\}$, e pertanto abbiamo

Base β	Nome rappresentazione	Cifre
2	Binaria	$d_i \in \{0,\ 1\}$
8	Ottale	$d_i \in \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}$
10	Decimale	$d_i \in \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9\}$
16	Esadecimale	$d_i \in \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A,\ B,\ C,\ D,\ E,\ F\}$

Il termine normalizzato sta quindi ad indicare che la parte intera del numero è zero e che non vi sono zeri iniziali nella mantissa, in quanto la virgola è stata spostata facendo assorbire tale operazione dall'esponente.

3.2 I numeri di macchina

I calcolatori utilizzano sempre una rappresentazione normalizzata per memorizzare al loro interno i numeri. Pertanto, fissata la base β adottata per la rappresentazione, un qualsiasi numero x viene memorizzato all'interno del calcolatore nella forma

segno	caratteristica	mantissa

e, quindi, un certo numero di posizioni di memoria (si può correttamente parlare di bit nel caso dell'aritmetica binaria) vengono destinate alla caratteristica, altre al segno ed altre ancora alla mantissa. In maniera molto intuitiva possiamo dire che la cartatteristica definisce la *grandezza del numero* mentre la mantissa rappresenta il *dettaglio del numero*.

Sebbene i numeri reali possano avere un numero infinito di cifre decimali, o avere in forma normalizzata un esponente E di qualsiasi grandezza, nel calcolatore sono ovviamente presenti dei limiti sia per le cifre della mantissa rappresentabili, sia per il minimo o massimo esponente, non potendo ipotizzare di lavorare su rappresentazioni non finite.

La retta reale pertanto non è tutta rappresentabile: avendo dei limiti sul valore della caratteristica non sarà possibile rappresentare numeri troppo grandi o troppo piccoli in valore assoluto (positivi o negativi); pertanto invece di avere una retta infinita, potremo rappresentare un intervallo su $\mathbb{R}$ la cui ampiezza dipende da quanto spazio destiniamo alla caratteristica.

All'interno di questo intervallo però, non tutti i numeri sono rappresentabili ma solo un sotto insieme finito; la numerosità di questo sottoinsieme dipende dal numero di cifre che utilizziamo per la mantissa. In generale tra un numero e l'altro che possiamo rappresentare, vi saranno infiniti numeri che non sono rappresentabili al calcolatore, come di seguito schematizzato.

.....|.........|.......|.....|...|.|||||.|...|.....|.......|.........|.....

$-\infty$ 0 $+\infty$

Pertanto, ad esempio, nel caso di una rappresentazione in base $\beta = 10$ con 4 cifre per la mantissa, si potrebbero rappresentare correttamente i numeri $0,3456$ e $0,3457$ ma non i numeri $0,34561$, $0,34562$, ecc. appartenenti all'intervallo $[0,3456\ ,\ 0,3457]$. Peraltro, se -8 e 8 sono gli estremi della caratteristica, allora $\pm 0,9999 \times 10^8$ sono i numeri in valore assoluto massimo che possiamo rappresentare, mentre $\pm 0,1 \times 10^{-8}$ è il più piccolo numero in forma normalizzata rappresentabile.

Indichiamo con $\mathbb{F}$ l'insieme, finito, dei numeri rappresentabili al calcolatore, detto **insieme dei numeri di macchina** ed osserviamo che tale insieme è caratterizzato da 4 costanti :

$$\begin{cases} \beta & : \quad \text{la base della rappresentazione} \\ t & : \quad \text{il numero di cifre della mantissa} \\ m & : \quad \text{il valore minimo per la caratteristica } E \\ M & : \quad \text{il valore massimo per la caratteristica } E \end{cases}$$

e pertanto possiamo scrivere $\mathbb{F} = \mathbb{F}(\beta, t, m, M)$. Qualsiasi numero di macchina $x \in \mathbb{F}(\beta, t, m, M)$ viene pertanto rappresentato in modo normalizzato come

$$x = \pm\, 0, d_1 d_2 \ldots d_t \;\times\; \beta^E \;,\; d_1 \neq 0 \tag{2}$$

e prende anche il nome di numero in **virgola mobile** o **floating point**.

Per memorizzare un numero di macchina sono necessari un certo numero di celle di memoria (che nel caso dell'aritmetica binaria comunemente utilizzata dai calcolatori sono i bit), che possiamo così schematizzare

$\pm$	E_1	E_2	$\ldots$	E_n	d_1	d_2	d_3	$\ldots$	$\ldots$	d_t

dove $E_1, E_2, \ldots, E_n$ sono i bit necessari per memorizzare la caratteristica, $d_1, d_2, \ldots, d_t$ sono i bit necessari per memorizzare la mantissa ed un ulteriore bit viene riservato al segno della mantissa.

Vediamo di seguito alcuni aspetti più meramente tecnici legati alla rappresentazione dei numeri al calcolatore.

Rappresentazione dello zero

Dal momento che si è imposto $d_1 \neq 0$, il valore zero non è rappresentabile in modo normalizzato e pertanto deve essere aggiunto per definizione all'insieme $\mathbb{F}(\beta, t, m, M)$. Vi sarà pertanto una combinazione di valori mediante la quale, in maniera convenzionale, si rappresenta lo zero.

Shifting e segno della caratteristica

Generalmente non viene utilizzata alcuna cella di memoria per il segno della caratteristica, in quanto si ricorre alla *rappresentazione in shifting della caratteristica*; dati, cioè, il valore minimo

m (di solito negativo) e massimo M (di solito positivo) che può assumere la caratteristica, ossia

$$m \leq E \leq M \ ,$$

il valore che viene effettivamente memorizzato è $E - m + 1$ e non E, in modo da avere valori che vanno da 1 a $M - m + 1$. Ad esempio, se l'esponente E può variare nell'insieme $[-126, 127]$ allora, per evitare di dover memorizzare il segno, l'esponente viene memorizzato come $\bar{E} = E - (-126) + 1 = E + 127$ e pertanto si memorizza il valore 1 per $E = -126$, il valore 2 per $E = -125$ e così via sino al valore 254 per $E = 127$.

Bit nascosto

Nel caso della rappresentazione binaria, quella più comunemente utilizzata all'interno dei calcolatori, dal momento che ciascuna cifra d_i assume i valori $\{0, 1\}$, dovendo essere per la normalizzazione $d_1 \neq 0$, e quindi $d_1 = 1$, generalmente viene omessa la cifra d_1. Pertanto con la rappresentazione binaria, di fatto, si utilizza invece della (2) una rappresentazione del tipo

$$x = \pm\,(1 + 0, d_1 d_2 \dots d_t) \times \ \beta^E = \pm\, 1, d_1 d_2 \dots d_t \ \times \ \beta^E \ ,$$

per cui il numero effettivo di cifre disponibili per la mantissa è $t + 1$ e non t.

Precisione, epsilon machine e cifre significative

Con il termine "*precisione*" di un sistema floating-point indichiamo il numero effettivo di cifre della mantissa. La precisione, che indichiamo con p, sarà in generale pari a t ma in un sistema con base $\beta = 2$, e che utilizza il bit nascosto, avremo $p = t + 1$.

Si definisce "*epsilon machine*" o anche "*unit roundoff*", e si indica mediante il simbolo ϵ, il più piccolo numero di macchina positivo che sommato ad 1 dia una quantità maggiore di 1. Si tratta quindi della distanza tra 1 ed il successivo numero di macchina. Come vedremo, l'espilon machine è importante per stimare l'errore che commesso ogni volta che si approssima con un numero di di macchina un numero che non appartiene ad $\mathbb{F}(\beta, t, m, M)$

Per determinare il valore di ϵ osserviamo che $\epsilon = (1 + \epsilon) - 1$. Pertanto, se p è la precisione, e quindi il numero di cifre effettivamente disponibili per la mantissa, otteniamo per sottrazione

$$\begin{array}{rcl}
1 + \epsilon & = & 0,1\underbrace{00\dots0}_{p-2}1 \times \beta^1 \\
1 & = & 0,1\underbrace{00\dots00}_{p-1} \times \beta^1 \\
\hline
\epsilon & = & 0,\underbrace{000\dots0}_{p-1}1 \times \beta^1
\end{array} \quad = \ 1 \times \beta^{1-p} \ = \ \beta^{1-p}$$

Esiste un modo semplice per calcolare l'epsilon machine. Essendo il più piccolo numero di macchina positivo ϵ tale che $1 + \epsilon > 1$, sarà sufficiente generare una successione di numeri di macchina u sempre più piccoli sino ad ottenere un valore per cui $1 + u = 1$. Il valore precedente sarà allora l'*epsilon machine* cercato. Per fare in modo che si generino effettivamente numeri di macchina, sarà sufficiente partire da 1 e dividere per la base della numerazione, in modo da

generare la successione dei numeri

$$\begin{aligned}
u_1 &= 1 = 0{,}1 \times \beta^1 \\
u_2 &= u_1/\beta = (0{,}1)_\beta = 0{,}1 \times \beta^0 = \frac{1}{\beta} \\
u_3 &= u_2/\beta = (0{,}01)_\beta = 0{,}1 \times \beta^{-1} = \frac{1}{\beta^2} \\
u_4 &= u_3/\beta = (0{,}001)_\beta = 0{,}1 \times \beta^{-2} = \frac{1}{\beta^3}
\end{aligned}$$

e così via (dal momento che i calcolatori utilizzano l'aritmetica binaria, chiaramente genereremo i numeri di macchina dividendo ripetutamente per 2). Un possibile algoritmo per il calcolo dell'*epsilon machine* è descritto in Algoritmo 1.

Algorithm 1 Calcolo precisione di macchina

```
1: u = 1.0
2: while (1 + u) > 1 do
3:     umem = u
4:     u = u/β
5: end while
6: u = umem
```

Una volta calcolato, mediante l'Algoritmo 1, l'*epsilon machine* possiamo ricavarci la precisione della macchina, ossia il numero di cifre effettivamente utilizzate per la mantissa. Essendo infatti $\epsilon = \beta^{1-p}$, abbiamo che

$$\log \epsilon = (1-p)\log \beta \iff p = 1 - \frac{\log \epsilon}{\log \beta}.$$

Strettamente correlato alla precisione è il concetto di *cifre significative*, ossia il numero di cifre corrette con le quali è possibile rappresentare un numero. In base β tale numero è chiaramente pari a p ma il più delle volte siamo interessati a conoscere il numero di cifre significative nella base decimale alla quale siamo maggiormente abituati.

Poiché la p-esima cifra risulterà l'ultima corretta nella base β per determinare l'ultima cifra corretta, che chiamiamo q nella base decimale sarà sufficiente eguagliare $\beta^{-p} \approx 10^{-q}$ da cui risolvendo otteniamo

$$q \approx -\log_{10} \beta^{-p} \approx p \log_{10} \beta.$$

3.3 Lo standard IEEE-754

Le moderne architetture dei calcolatori sono progettate in base binaria (quindi $\beta = 2$) e prevedono due diverse modalità di rappresentare i numeri: la rappresentazione dei numeri in **singola precisione** che richiede 32 bit (e quindi 4 byte) e quella in **doppia precisione** per la quale sono necessari 64 bit (e quindi 8 byte). Esiste anche la cosiddetta **mezza precisione** la quale richiede 16 bit (e quindi 2 byte) e viene utilizzata per lo più nei processori grafici per risparmiare l'occupazione di memoria.

Come questi bit vengano ripartiti tra mantissa e caratteristica dipende dalla macchina; una delle configurazioni di riferimento, codificata nello standard IEEE (Institute of Electrical and

Electronic Engineers) n. 754, ed usata nella maggior parte dei calcolatori di nuova generazione (in particolare sui personal computer), è la seguente

Precisione	Segno	Caratteristica	Mantissa	Tot.
		c	t	
Mezza	1	5	10	16
Singola	1	8	23	32
Doppia	1	11	52	64

Sulla base di questa tabella è possibile determinare i numeri rappresentabili al calcolatore in una determinata precisione (ci limitiamo di seguito ad analizzare il caso della singola precisione ma un discorso analogo può essere fatto per la doppia e per la mezza precisione che si lascia come esercizio per lo studente).

Valori rappresentabili in singola precisione

Avendo nella singola precisione 8 cifre per l'esponente si possono rappresentare esponenti nell'intervallo da 0 a $2^8 - 1$, ossia da 0 a 255; eliminando gli estremi 0 e 255, solitamente riservati ad altri scopi (quali la rappresentazione dello zero, dell'underflow e dell'overflow e di altri valori non validi), e tenendo conto della rappresentazione del segno dell'esponente mediante shifting, gli effettivi valori che potrà assumere l'esponente sono compresi tra $m = -126$ e $M = +127$.

Per quanto riguarda la mantissa, invece abbiamo a disposizione tutte le $t+1$ cifre (tenendo conto del bit nascosto) per valori che andranno da

$$R_{min} = 1,\underbrace{000000\ldots00}_{t \text{ zeri}} \quad \text{a} \quad R_{max} = 1,\underbrace{111111\ldots11}_{t \text{ cifre}}.$$

Il valore R_{min} della più piccola mantissa è evidentemente pari ad 1. Per quanto riguarda il valore più grande R_{max} che può assumere la mantissa invece esso è un valore prossimo a 2 ma immediatamente precedente a 2. Questo valore sarebbe pari a 2 se incrementassimo la sua ultima cifra, e quindi

$$2 = 1,\underbrace{111111\ldots11}_{t \text{ cifre}} + 0,\underbrace{000000\ldots0}_{t-1 \text{ zeri}}1 = R_{max} + 2^{-t}$$

da cui

$$R_{max} = 2 - 2^{-t}.$$

Grazie a queste informazioni siamo adesso in grado di stabilire quale sia il più piccolo e il più grande numero di macchina (in valore assoluto) rappresentabile al calcolatore. Tali valori, che prendono rispettivamente il nome di *RealMin* e *RealMax*, sono

$$\mathit{RealMin} = 1 \times 2^m = 1 \times 2^{-126} \approx 1,17549 \times 10^{-38}$$

e

$$\mathit{RealMax} = \left(2 - 2^{-t}\right) \times 2^M = \left(2 - 2^{-23}\right) \times 2^{127} \approx 3,40282 \times 10^{38}.$$

L'epsilon machine in singola precisione è quindi $\epsilon = \beta^{1-p} = 2^{-23} \approx 1.1921 \times 10^{-7}$ mentre il numero di cifre significative è dato da $q \approx p \log_{10} 2 = 24 \log_{10} 2 \approx 7,22$.

Nella tabella seguente si riassumono le informazioni principali relative ai numeri di macchina rappresentabili nelle diverse precisioni previste dallo standard IEEE-754, lasciando allo studente il compito di ricavare e verificare tali informazioni.

Precis.	m	M	p	*RealMin*	*RealMax*	ϵ
Mezza	-14	15	11	$1,0 \times 2^{-14}$ $\approx 6,1035 \times 10^{-5}$	$(2 - 2^{-10}) \times 2^{15}$ $\approx 6,5504 \times 10^{4}$	$2^{-10} \approx$ $9,7656 \times 10^{-4}$
Singola	-126	127	24	1.0×2^{-126} $\approx 1,1755 \times 10^{-38}$	$(2 - 2^{-23}) \times 2^{127}$ $\approx 3,4028 \times 10^{38}$	$2^{-23} \approx$ $1,1921 \times 10^{-7}$
Doppia	-1022	+1023	53	$1,0 \times 2^{-1022}$ $\approx 2,2251 \times 10^{-308}$	$(2 - 2^{-52}) \times 2^{1023}$ $\approx 1,7977 \times 10^{308}$	$2^{-52} \approx$ $2,2204 \times 10^{-16}$

Su alcune architetture si affaccia anche la **quadrupla precisione** la quale utilizza 16 byte, per un totale di $16 * 8 = 128$ bit così ripartiti: 15 bit per la caratteristica e 112 bit per la mantissa, oltre naturalmente al bit destinato al segno.

Esempio 3.1 (Rappresentazione al calcolatore del numero decimale 0,1)**.** E' facile verificare che il numero decimale $(0{,}1)_{10}$ equivale in aritmetica binaria al numero periodico $(0{,}000\overline{1100})_2 = (0{,}\overline{1100} \times 2^{-3})_2$ per cui, evitando di memorizzare la prima cifra della mantissa che deve essere sempre pari ad 1, abbiamo $(1{,}100\overline{1100} \times 2^{-4})_2$.

Con l'aritmetica in singola precisione e con il rounding (che studieremo a breve), secondo lo standard IEEE-754 le cifre da rappresentare per il valore $(0{,}1)_{10}$ sono

0

segno (positivo) della mantissa

0	1	1	1	1	0	1	1

caratteristica $\bar{E} = E - (-126) + 1 = 123$ essendo $E = -4$

1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1

mantissa

che vengono impacchettati nei quattro byte

00111101	11001100	11001100	11001101

costituenti un numero in singola precisione.

Dal momento che la mantissa (completa della cifra 1 iniziale) 1,10011001100110011001101 in notazione decimale equivale a 1,60000002, abbiamo che la rappresentazione al calcolatore in singola precisione del valore decimale 0,1 è data da

$$fl(0{,}1) = +1{,}600000023841858 \times 2^{-4} = 0{,}100000001490116 \neq 0,1$$

che differisce dal valore di partenza. Tale differenza è dovuta al fatto che 0,1 ha in binario una rappresentazione periodica che non è possibile memorizzare in maniera esatta all'interno del calcolatore.

Con l'aritmetica in doppia precisione, invece, essendo $m = 1022$, $M = 1023$ e $t = 52$ la caratteristica $E = -4$, da memorizzare in forma shiftata, diventa

$$\bar{E} = E - m + 1 = -4 - (-1022) + 1 = (1019)_{10} = (1111111011)_2$$

mentre per la mantissa, utilizzando il rounding, si ha

$$d_1 d_2 \dots d_{52} = 1001100110011001100110011001100110011001100110011010$$

e quindi i valori da memorizzare sono

0	segno della mantissa

0	1	1	1	1	1	1	1	0	1	1	caratteristica

1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	mantissa
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	

che devono essere impacchettati negli otto byte che formano un numero in doppia precisione, per cui si ha che il valore $(0,1)_{10}$ viene rappresentato al calcolatore come

1 :	00111111	2 :	10111001	3 :	10011001	4 :	10011001
5 :	10011001	6 :	10011001	7 :	10011000	8 :	10011001

00111111 10111001 10011001 10011001 10011001 10011001 10011001 10011010

Per ricostruire il numero rappresentato, osserviamo che la mantissa (completa del valore uno iniziale) equivale in decimale a

$$(1{,}600000001)_{10}$$

e pertanto la rappresentazione floating point in doppia precisione di 0,1 è data da

$$\begin{aligned} fl(0{,}1) &= +1{,}600000001 \times 2^{-4} \\ &= 0{,}1000000000000000055511151231257827021181583404541015625 \neq 0{,}1 \end{aligned}$$

che rappresenta sicuramente una migliore approssimazione rispetto a quella ottenuta in semplice precisione, ma pur sempre affetta da errore.

3.4 Overflow e underflow

Con la rappresentazione dei numeri appena descritta possono sorgere alcuni problemi:

- non si può rappresentare un numero in valore assoluto più grande di quello massimo; un eventuale tentativo di memorizzare un numero x con caratteristica E maggiore del massimo consentito M produrrebbe come effetto un errore detto di **overflow**;
- non si può rappresentare un numero in valore assoluto più piccolo del minimo rappresentabile; l'eventuale tentativo di memorizzare un numero x con caratteristica E più piccola di m produrrebbe un errore detto di **underflow** e solitamente viene restituito il valore zero pur in presenza di valori diversi da zero; in sostanza il calcolatore non è in grado di distinguere valori piccoli oltre una certa soglia e li considera tutti uguali a zero.

Naturalmente anche queste situazioni sono rappresentate in un qualche modo all'interno del calcolatore. Nella rappresentazione IEEE-754 in singola precisione questi valori sono rappresentati come

0 11111111 00000000000000000000000	$(7F800000)_{16}$	Overflow positivo $= +\infty$
1 11111111 00000000000000000000000	$(FF800000)_{16}$	Overflow negativo $= -\infty$
0 00000000 00000000000000000000000	$(00000000)_{16}$	Underflow positivo $= +0.0$
1 00000000 00000000000000000000000	$(80000000)_{16}$	Underflow negativo $= -0.0$
0 11111111 10000000000000000000000	$(7FC00000)_{16}$	Not a Number $= NaN$

mentre tutti i valori per con caratteristica 00000000 e mantissa con almeno un bit diverso da zero vengono utilizzati, in alcune architetture, per rappresentare numeri più piccoli del minimo consentito (anche se con precisione ridotta) mediante una rappresentazione denormalizzata.

3.5 Trasformazione di numeri reali in numeri floating-point: chopping e rounding

Sia a causa del numero limitato di cifre a disposizione per la mantissa, che per il limite inferiore m e superiore M che deve soddisfare la caratteristica, non tutti i numeri reali possono essere rappresentati al calcolatore.

Nel caso di numeri troppo grandi o troppo piccoli, quali quelli con caratteristica fuori dal range $[m, M]$, come abbiamo visto, non vi sono particolari soluzioni e ci si limita a segnalare una situazione di overflow e ad interrompere l'esecuzione se $E > M$ o a segnalare l'underflow e sostituire al numero molto piccolo il valore 0 nel caso di underflow.

Nel caso invece di numeri in cui la mantissa presenta un numero elevato di cifre, dal momento che esistono comunque numeri di macchina piuttosto vicini, si può pensare di sostituire al numero reale uno dei numeri di macchina che meglio approssimano il numero reale non rappresentabile.

Dato un numero reale $x \in \mathbb{R}$ che non appartenga anche a $\mathbb{F}$ (è, ad esempio, il caso del valore decimale $(0,1)_{10}$ che nella rappresentazione binaria ha un numero infinito di cifre), è necessario avere a disposizione una qualche funzione, che indichiamo genericamente con $fl(x)$,

$$fl(x) : \mathbb{R} \to \mathbb{F}(\beta, t, m, M)$$

che associ ad x uno tra i numeri floating point $\tilde{x} \in \mathbb{F}$ a disposizione. Sono essenzialmente due le funzioni utilizzate nei calcolatori per effettuare questa associazione.

Chopping (*round towards zero*)

Consideriamo un numero x avente in base β una rappresentazione del tipo

$$x = \pm\, 0, d_1 d_2 \dots d_p d_{p+1} d_{p+2} \cdots \times \beta^E,$$

(evidentemente x non fa parte di $\mathbb{F}(\beta, t, m, M)$ in quanto la sua mantissa ha un numero di cifre superiore alla precisione p) allora la funzione di "*chopping*", denotata con $chop(x)$, considera solo le prime p cifre ed ignora le cifre dalla $p+1$-esima in poi. Pertanto il numero di macchina $\tilde{x}$ corrispondente a x per mezzo della funzione di troncamento è dato da

$$\tilde{x} = chop(x) = \pm\, 0, d_1 d_2 \dots d_p \times \beta^E\ .$$

In questo caso è facile calcolare l'errore assoluto come

$$e_A(chop(x)) = |x - chop(x)| = 0,\underbrace{00\ldots0}_{p\text{-volte}}d_{p+1}d_{p+2}\cdots \times \beta^E = 0,d_{p+1}d_{p+2}\cdots \times \beta^{E-p} < \beta^{E-p}$$

mentre per l'errore relativo, essendo $|x| \geq 0,d_1 \times \beta^E = d_1 \times \beta^{E-1} \geq \beta^{E-1}$ dal momento che $d_1 \neq 0$, abbiamo che

$$e_R(chop(x)) = \left|\frac{x - chop(x)}{x}\right| < \frac{\beta^{E-p}}{\beta^{E-1}} = \beta^{1-p}\ .$$

Notiamo come l'errore relativo, a differenza di quello assoluto, non dipende dalla grandezza del numero ma solo dal numero di cifre della mantissa. Pertanto, tante più sono le cifre memorizzabili nella mantissa e tanto più piccolo sarà l'errore relativo che commetteremo.

L'errore assoluto invece dipende dalla sua caratteristica; non possiamo dare una sua maggiorazione a priori, ma osserviamo come l'errore assoluto sia tanto maggiore quanto più grande è il valore di E. Questo fenomeno è compatibile col fatto che, come abbiamo osservato, i numeri in $\mathbb{F}$ sono tanto più densi per valori piccoli e tanto più radi per valori grandi. Pertanto, quando si deve rappresentare un numero molto grande (e quindi con E grande), essendo grande la distanza tra i due numeri di $\mathbb{F}$ che comprendono il dato x, tanto più grande potrà essere l'errore che commetteremo nel sostituire x con uno di questi due numeri.

Il chopping viene denominato anche come troncamento o *round towards zero*" in quanto sia nel caso positivo che negativo si approssima con un numero più vicino allo zero rispetto al dato.

Per quanto riguarda lo standard IEEE-754, sulla base dei risultati appena visti, diamo nella tabella seguente una maggiorazione per l'errore relativo introdotto dall'operazione di chopping nelle diverse precisioni.

	p	$e_R(chop(x)) < \beta^{1-p}$
Singola precisione	24	$< 2^{-23} \approx 1,1920928... \times 10^{-7}$
Doppia precisione	53	$< 2^{-52} \approx 2,2204460... \times 10^{-16}$

Rounding (*round to nearest*)

Attraverso il *rounding* si punta ad associare ad ogni numero x il numero di macchina $\tilde{x}$ più vicino. Per semplicità consideriamo un numero x di segno positivo

$$x = 0,d_1d_2\ldots d_pd_{p+1}d_{p+2}\cdots \times \beta^E,$$

ed individuiamo i due numeri di macchina $\tilde{x}_1$ e $\tilde{x}_2$ più vicini ad x e quindi tali che $\tilde{x}_1 \leq x \leq \tilde{x}_2$ ($\tilde{x}_1$ è quindi il più grande tra i numeri di macchina minori o uguali a x ed $\tilde{x}_2$ il più piccolo tra i numeri di macchina maggiori o uguali a x). Questi numeri sono dati da

$$\tilde{x}_1 = 0,d_1d_2\ldots d_p \times \beta^E$$

e dal numero successivo a $\tilde{x}_1$, ottenuto aumentando quindi di 1 l'ultima cifra (la p-esima) di $\tilde{x}_1$. Poiché il peso della p-esima cifra di un numero (trascurando la parte esponente) è dato da β^{-p}, per aumentare di 1 la p-esima cifra sarà sufficiente sommare la quantità β^{-p} per cui abbiamo

$$\tilde{x}_2 = (0,d_1d_2\ldots d_p + \beta^{-p}) \times \beta^E = 0,d_1d_2\ldots d_p \times \beta^E + \beta^{-p} \times \beta^E = \tilde{x}_1 + \beta^{E-p}.$$

Il rounding di x associa al numero x uno dei numeri $\tilde{x}_1$ o $\tilde{x}_2$ a seconda di quale risulta essere più vicino ad x. Per effettuare questo confronto si considera il punto medio tra $\tilde{x}_1$ e $\tilde{x}_2$, dato da

$$\begin{aligned}\bar{x} &= \frac{\tilde{x}_1 + \tilde{x}_2}{2} = \frac{\tilde{x}_1 + \tilde{x}_1 + \beta^{E-p}}{2} = \tilde{x}_1 + \frac{1}{2}\beta^{E-p} \\ &= (0, d_1 d_2 \dots d_p + \frac{1}{2}\beta^{-p}) \times \beta^E\end{aligned}$$

e si effettua la seguente scelta

$$\text{se } x < \bar{x} \text{ e quindi } x \in [\tilde{x}_1, \bar{x}) \quad : \quad round(x) = \tilde{x}_1$$

$$\text{se } x \geq \bar{x} \text{ e quindi } x \in [\bar{x}, \tilde{x}_2] \quad : \quad round(x) = \tilde{x}_2$$

Dal punto di vista pratico per effettuare questo confronto è sufficiente guardare la $p+1$-esima cifra di x e se essa è inferiore a $\frac{1}{2}\beta$ si effettua un troncamento, mentre se è maggiore o uguale a $\frac{1}{2}\beta$ si aumenta di una unità la p-esima cifra, secondo il seguente schema

$$\text{se } d_{p+1} < \frac{\beta}{2} \text{ e quindi } x < \bar{x} \quad : \quad round(x) = \tilde{x}_1$$

$$\text{se } d_{p+1} \geq \frac{\beta}{2} \text{ e quindi } x \geq \bar{x} \quad : \quad round(x) = \tilde{x}_2$$

Con il rounding l'errore assoluto risulta essere maggiorata da

$$e_A(round(x)) = |x - round(x)| \leq |\bar{x} - round(x)| \leq \frac{1}{2}\beta^{E-p}$$

mentre, ricordando che $|x| \geq \beta^{E-1}$, la maggiorazione per l'errore relativo è data da

$$e_R(round(x)) = \left|\frac{x - round(x)}{x}\right| \leq \frac{1}{2}\frac{\beta^{E-p}}{\beta^{E-1}} = \frac{1}{2}\beta^{1-p} \; .$$

Il rounding pertanto ci permette di introdurre un errore che risulta in molti casi più piccolo di quello introdotto con il chopping.

L'operazione di rounding viene definita anche come "*round to nearest*" in quanto approssima verso il numero di macchina più vicino.

Errori ed Epsilon machine

Sia che si utilizzi il chopping che il rounding (che indichiamo da ora in poi genericamente come operazione di floating $fl(x)$), l'errore relativo può essere descritto come

$$e_R(fl(x)) = \left|\frac{x - fl(x)}{x}\right| \leq u, \tag{3}$$

con la quantità u data da

$$u = \begin{cases} \beta^{1-p} = \epsilon & \text{se si utilizza il chopping} \\ \frac{1}{2}\beta^{1-p} = \frac{1}{2}\epsilon & \text{se si utilizza il rounding} \end{cases} .$$

Il più delle volte si utilizza una combinazione di rounding e chopping, preferendo nella quasi totalità dei casi il rounding, ricorrendo però al chopping per quei numeri maggiori del realmax positivo (o minori del realmax negativo) che, se approssimati col rounding ricadrebbero nell'overflo, mentre se approssimati col chopping danno ancora un numero di macchina.

Dalla formula dell'errore relativo (3) possiamo ottenere una relazione che lega ciascun numero reale alla sua rappresentazione floating-point ponendo

$$\delta_x = \frac{fl(x) - x}{x}, \quad |\delta_x| \leq u.$$

Possiamo pertanto riscrivere in forma generale la funzione $fl(x)$ come

$$fl(x) : \mathbb{R} \to \mathbb{F}(\beta, t, m, M), \quad fl(x) = x(1 + \delta_x) = x + x\delta_x, \quad |\delta_x| \leq u$$

dove quindi $x\delta_x$ è la perturbazione introdotta sul numero dalla conversione in floating-point e δ_x il corrispondente errore relativo. La maggiorazione u dell'errore relativo dipende da tre fattori:

- base β della rappresentazione;
- numero di cifre t utilizzate per rappresentare la mantissa (e quindi precisione p);
- tipo di trasformazione floating-point adottata (chopping o rounding).

3.6 Python ed i numeri di macchina

Python dispone di diversi comandi per esplorare i numeri di macchina di cui dispone il nostro calcolatore. In particolare, una volta richiamato il modulo `numpy` mediante l'istruzione

```
>>> import numpy as np
```

è possibile utilizzare i seguenti comandi per ottenere alcune importanti informazioni.

Comando	**Significato**
`np.finfo(np.float).eps`	Precision machine
`np.finfo(np.float).tiny`	Real min
`np.finfo(np.float).max`	Real max
`np.finfo(np.float).nmant`	N.ro cifre per la mantissa
`np.finfo(np.float).nexp`	N.ro cifre per la caratteristica
`np.finfo(np.float).maxexp`	Più piccola caratteristica che causa overflow $(M+1)$
`np.finfo(np.float).minexp`	Più piccola caratteristica (m)

Per un elenco di tutte le possibili informazioni che possiamo ottenere sui numeri di macchina mediante il comando `np.finfo` si utilizzi la guida in linea `help(np.finfo)`.

E' possibile inoltre ottenere le stesse informazioni relativamente alla mezza e singola precisione sostituendo `np.float` con `np.half` e `np.single` rispettivamente.

Sempre attraverso il modulo `numpy` è possibile effettuare operazioni in singola e mezza precisione. Ad esempio le istruzioni `U = np.float32(0.1)` e `np.float32(U + 0.2)` consentono, rispettivamente, di effettuare una assegnazione (inizializzando la corrispondente variabile) in singola precisione e di effettuare una somma in singola precisione. Analogamente per operare in mezza precisione è sufficiente sostituire `np.float32` con `np.float16`.

3.7 L'aritmetica dei calcolatori

La conversione dei numeri reali in numeri floating-point non introduce errori solo sui dati ma anche sui risultati delle operazioni, portando il calcolatore ad operare in una aritmetica differente da quella reale che viene appunto definita come *aritmetica dei calcolatori*. Per meglio illustrare le differenze tra aritmetica reale ed aritmetica dei calcolatori è utile partire da un semplice esempio.

Esempio 3.2. Consideriamo un calcolatore con aritmetica $\mathbb{F}(10, 5, -10, 10)$, cioè in base $\beta = 10$ e con $t = 5$ cifre per la rappresentazione della mantissa, per mezzo del quale vogliamo effettuare la somma dei due numeri di macchina

$$\tilde{x} = 145{,}67 = 0{,}14567 \times 10^3 \qquad \tilde{y} = 2{,}3214 = 0{,}23214 \times 10^1.$$

Per poter essere sommati i due numeri devono essere incolonnati e rappresentati con la stessa parte esponente

$$\begin{array}{rcccl} \tilde{x} & = & 0{,}14567 \times 10^3 & = & 0{,}14567 \qquad \times 10^3 \\ \tilde{y} & = & 0{,}23214 \times 10^1 & = & 0{,}0023214 \times 10^3 \\ \hline \tilde{x} + \tilde{y} & = & & = & 0{,}1479914 \times 10^3 \end{array}$$

(operazione possibile nei comuni calcolatori i quali, per effettuare le operazioni, utilizzano registri di memoria più ampi). Il risultato $\tilde{x} + \tilde{y}$ così ottenuto ha però un numero di cifre della mantissa maggiore di t e quindi non è un numero di macchina. Il calcolatore dovrà pertanto associare a tale numero uno dei numeri di $\mathbb{F}(10, 5, -10, 10)$. Sia che si utilizza il *chopping* o il *rounding*, in questo caso il risultato ottenuto è

$$fl(\tilde{x} + \tilde{y}) = 0{,}14799 \times 10^3 = 147{,}99$$

e quindi, pur partendo da numeri di $\mathbb{F}(10, 5, -10, 10)$ si è introdotto un errore sul risultato rispetto al valore teorico di $x + y = 147{,}9917$.

L'esempio precedente ci ha mostrato come, pur avendo a disposizione due numeri di macchina $\tilde{x}, \tilde{y} \in \mathbb{F}$, la loro somma $\tilde{x} + \tilde{y}$ può non appartenere a $\mathbb{F}$ e pertanto la conversione del risultato in un numero di macchina può introdurre un errore.

A differenza di $\mathbb{R}$, l'insieme $\mathbb{F}$ non è chiuso rispetto all'operazione di somma

$$\textbf{non chiusura di } \mathbb{F} \textbf{ rispetto a } + : \quad \tilde{x}, \tilde{y} \in \mathbb{F} \not\Longrightarrow \tilde{x} + \tilde{y} \in \mathbb{F}$$

e la mancanza d di chiusura è purtroppo comune alla maggior parte delle operazioni.

Anche la rappresentazione, mediante uno dei numeri di macchina, del risultato di $\tilde{x} + \tilde{y}$ può essere descritta come

$$fl\big(\tilde{x} + \tilde{y}\big) = \big(\tilde{x} + \tilde{y}\big)\big(1 + \delta\big), \quad |\delta| \leq u,$$

dove δ è l'errore relativo che si introduce.

In generale possiamo descrivere il risultato di una operazione di macchina come il risultato esatto dell'operazione a cui viene applicato il chopping o rounding. Se indichiamo pertanto con il simbolo $\diamond$ una generica operazione in aritmetica reale (e quindi con precisione infinita), e con il simbolo $\circleddiamond$ la corrispondente operazione di macchina (quella effettuata in aritmetica con precisione finita), allora abbiamo

$$\tilde{x} \circleddiamond \tilde{y} = fl(\tilde{x} \diamond \tilde{y}) = (\tilde{x} \diamond \tilde{y})(1 + \delta) \;\text{ con }\; |\delta| \leq u \,,$$

dove $\tilde{x}$ e $\tilde{y}$ sono numeri floating-point. Questo tipo di aritmetica prende il nome di **aritmetica di macchina** o **aritmetica in virgola mobile**.

L'aritmetica di macchina si comporta in modo differente rispetto all'aritmetica reale e molte delle proprietà dell'aritmetica reale non sono purtroppo più valide nell'aritmetica dei calcolatori. Elenchiamo qui alcune classiche proprietà dell'aritmetica reale che in generale non risultano più valide inb aritmetica dei calcolatori:

Associatività della somma	$a + (b + c) = (a + b) + c$
Associatività del prodotto	$a(bc) = (ab)c$
Distributività del prodotto rispetto alla somma	$a(b + c) = ab + ac$
Legge di cancellazione della somma	$a + b = a + c \Rightarrow b = c$
Legge di cancellazione del prodotto	$ab = ac,\ a \neq 0 \Rightarrow b = c$
Legge di semplificazione	$a(b/a) = b$

Esempio 3.3. [Non associatività della somma] Consideriamo, per semplicità, una aritmetica in virgola mobile del tipo $\mathbb{F}(10, 3, m, M)$, cioè in base $\beta = 10$ e con $t = 3$ cifre per la rappresentazione della mantissa, e che utilizzi il chopping. Supponiamo di avere i seguenti numeri reali ed i corrispondenti numeri di macchina in $\mathbb{F}(10, 3, m, M)$

$$\begin{cases} x = 23,242 & : \ \tilde{x} = 0,232 \times 10^2 \\ y = 1,232 & : \ \tilde{y} = 0,123 \times 10^1 \\ z = 2,383 & : \ \tilde{z} = 0,238 \times 10^1 \end{cases} .$$

Il risultato teorico, sia che si calcoli $x + (y + z)$ che $(x + y) + z$ è, $26,857 = 0,26857 \times 10^2$. Possiamo quindi calcolare mediante l'aritmetica del nostro calcolatore

$$\begin{aligned} x \oplus (y \oplus z) &= 0,232 \times 10^2 \oplus (0,123 \times 10^1 \oplus 0,238 \times 10^1) = \\ &= 0,232 \times 10^2 \oplus 0,361 \times 10^1 = 0,232 \times 10^2 \oplus 0,0361 \times 10^2 = \\ &= 0,268 \times 10^2 = 26,8 \\ (x \oplus y) \oplus z &= (0,232 \times 10^2 \oplus 0,123 \times 10^1) \oplus 0,238 \times 10^1 = \\ &= (0,232 \times 10^2 \oplus 0,0123 \times 10^2) \oplus 0,238 \times 10^1 = 0,2443 \times 10^2 \oplus 0,238 \times 10^1 = \\ &= 0,244 \times 10^2 \oplus 0,238 \times 10^1 = 0,244 \times 10^2 \oplus 0,0238 \times 10^2 = \\ &= 0,2678 \times 10^2 = 0,267 \times 10^2 = 26,7 \end{aligned}$$

e pertanto abbiamo che $x \oplus (y \oplus z) \neq (x \oplus y) \oplus z$.

Esempio 3.4 (Mancata validità della legge di cancellazione della addizione)**.** Si provi, su un normale calcolatore, ad immettere i dati

$$a = 1.0 \times 10^{20}, \quad b = 1.0 \times 10^{-12}, \quad c = 1.0 \times 10^{-11}$$

e si verifichi che $a + b = a + c$ nonostante sia $b \neq c$.

Esempio 3.5 (Differenza tra aritmetica reale ed aritmetica dei calcolatori)**.** Il seguente esempio ci permette di osservare come semplici operazioni effettuate al calcolatore non sempre diano i risultati attesi. Ad esempio, posto $a = 3.0 \times 10^{-16}$ e $b = 1.0$, si verifica, eseguendo la semplice istruzione, che al calcolatore il valore di

$$-1 + (b + a)$$

risulta differente dal valore atteso a e che, in questo caso, solo una cifra decimale risulti corretta. Diversamente, se invece di calcolare $-1+(b+a)$, avessimo organizzato il calcolo come $(-1+b)+a$ non ci sarebbero stati problemi. Analogamente si può verificare che l'operazione

$$(-1 + (b + a))/a$$

non restituisce il risultato 1 che invece ci attenderemmo eseguendo il calcolo in aritmetica reale.

Si verifichino questi esempi fornendo invece un valore del tipo $a = 2^{-51}$ e si provi a spiegare il motivo del diverso comportamento del calcolatore.

3.8 Propagazione di errori: errore inerente, algoritmico e totale

Nelle sezioni precedenti abbiamo potuto notare che nell'effettuare un'operazione riscontriamo la presenza di due tipi di errori: un errore sui dati (dovuto ad errori di misurazione ma anche alla conversione dei dati in numeri floating-point) ed un errore sul risultato (legato alla conversione del risultato dell'operazione in numero floating-point).

Il primo errore prende il nome di *errore inerente* ed il secondo di *errore algoritmico*. Dobbiamo studiare come questi due errori interagiscono e l'effetto prodotto sul risultato finale.

Per semplificare supporremo che i dati siano in partenza esatti e che l'errore inerente sia dunque solo quello introdotto dalla rappresentazione dei dati in virgola mobile.

A tal proposito consideriamo due numeri reali $x, y \in \mathbb{R}$ e che vogliamo calcolare una generica operazione $x \diamond y$. Se x e y non sono numeri di macchina (quindi $x, y \notin \mathbb{F}$) per prima cosa il calcolatore dovrà convertire x e y in numeri floating-point $\tilde{x}, \tilde{y} \in \mathbb{F}$, mediante una delle operazioni a disposizione (ad esempio, *chopping* o *rounding*).

Pertanto i dati del problema subiscono una perturbazione

$$\begin{cases} \tilde{x} = x(1 + \delta_x) \\ \tilde{y} = y(1 + \delta_y) \end{cases} \quad \text{con} \ \ |\delta_x|, |\delta_y| \leq u$$

e se anche l'operazione $\diamond$ venisse effettuata in aritmetica reale darebbe comunque un errore dal momento che i dati sono perturbati. Si tratta dell'*errore inerente* che è quindi dato da

$$E_{in} = \frac{|(x \diamond y) - (\tilde{x} \diamond \tilde{y})|}{|x \diamond y|}.$$

Poiché però l'operazione viene effettuata al calcolatore, e non è detto che $\mathbb{F}$ sia chiuso rispetto a $\diamond$ (di solito non lo è), per poter immettere il risultato in memoria, anche $\tilde{x} \diamond \tilde{y}$ dovrà essere convertito in numero di macchina, restituendo quello che denotiamo come $\tilde{x} \circleddiamond \tilde{y}$ rinveniente dalla perturbazione di $\tilde{x} \diamond \tilde{y}$

$$\tilde{x} \circleddiamond \tilde{y} = fl(\tilde{x} \diamond \tilde{y}) = (\tilde{x} \diamond \tilde{y})(1 + \delta) \ \ \text{con} \ \ |\delta| \leq u$$

e l'errore che questa perturbazione comporta è l'*errore algoritmico*, che è quindi dato da

$$E_{alg} = \frac{|(\tilde{x} \diamond \tilde{y}) - (\tilde{x} \circleddiamond \tilde{y})|}{|\tilde{x} \diamond \tilde{y}|}.$$

Riassumiamo la catena dei due errori mediante il seguente schema

$$\begin{matrix} x \in \mathbb{R} \\ y \in \mathbb{R} \end{matrix} \xrightarrow[\text{Err. inerente}]{} \begin{matrix} \tilde{x} = fl(x) \in \mathbb{F} \\ \tilde{y} = fl(y) \in \mathbb{F} \end{matrix} \xrightarrow[\text{Operazione}]{} \tilde{x} \diamond \tilde{y} \in \mathbb{R} \xrightarrow[\text{Err. algoritmico}]{} fl(\tilde{x} \diamond \tilde{y}) \in \mathbb{F}$$

I due errori, l'errore inerente e l'errore algoritmo, sono studiati in modo differente.

Abbiamo già visto che l'errore inerente è legato al *condizionamento* del problema: in un problema ben condizionato a piccoli errori sui dati corrispondono piccoli errori sui risultati, altrimenti il problema è mal condizionato.

L'errore algoritmico invece dipende da come effettuiamo le operazioni (ad esempio dal loro ordine) e quindi dall'algoritmo utilizzato e per questo si introduce il concetto di *stabilità*. Un algoritmo si dice *stabile* se non amplifica molto l'errore, altrimenti si dice *instabile*.

Condizionamento e stabilità ci dicono entrambi come gli errori si amplificano. Il condizionamento però è una caratteristica intrinseca del problema da risolvere, la stabilità invece dell'algoritmo utilizzato per risolvere il problema.

Poiché il risultato atteso è $x \diamond y$, mentre quello che effettivamente otterremo al calcolatore sarà $\tilde{x} \circledcirc \tilde{y}$, il corrispondente errore relativo è dato da

$$e_R(x \diamond y) = \frac{|x \diamond y - \tilde{x} \circledcirc \tilde{y}|}{|x \diamond y|}.$$

Per valutare questo errore lo riscriviamo in termini dell'errore inerente e dell'errore algoritmico dal momento che

$$\begin{aligned} e_R(\tilde{x} \circledcirc \tilde{y}) &= \left| \frac{(x \diamond y) - (\tilde{x} \circledcirc \tilde{y})}{x \diamond y} \right| = \\ &= \left| \frac{(x \diamond y) - (\tilde{x} \diamond \tilde{y})}{x \diamond y} + \frac{(\tilde{x} \diamond \tilde{y}) - (\tilde{x} \circledcirc \tilde{y})}{x \diamond y} \right| \leq \\ &\leq \left| \frac{(x \diamond y) - (\tilde{x} + \tilde{y})}{x \diamond y} \right| + \left| \frac{(\tilde{x} \diamond \tilde{y}) - (\tilde{x} \circledcirc \tilde{y})}{\tilde{x} \diamond \tilde{y}} \right| \left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} \right| \\ &= E_{in} + E_{alg} \cdot \left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} \right|. \end{aligned}$$

Per studiare il fattore che moltiplica l'errore algoritmico osserviamo che

$$\left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} \right| = \left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} - 1 + 1 \right| \leq \left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} - 1 \right| + 1 = \left| \frac{\tilde{x} \diamond \tilde{y} - x \diamond y}{x \diamond y} \right| + 1 = E_{in} + 1$$

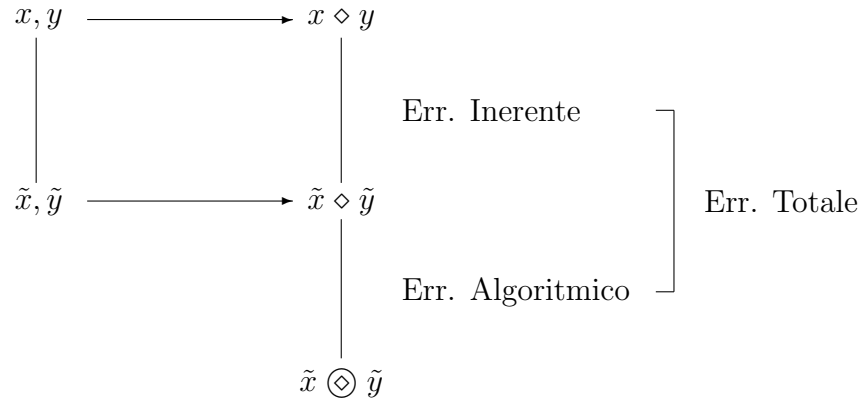
e quindi possiamo maggiorare

$$\left| \frac{\tilde{x} \diamond \tilde{y}}{x \diamond y} \right| \leq E_{in} + 1$$

da cui otteniamo

$$e_R(\tilde{x} \circledcirc \tilde{y}) \leq E_{in} + E_{alg}\big(E_{in} + 1\big) = E_{in} + E_{alg} + E_{in}E_{alg} \approx E_{in} + E_{alg},$$

dove abbiamo semplificato eliminando il termine $E_{in}E_{alg}$ dal momento che dovendo essere E_{in} e E_{alg} piccoli, è ragionevole assumere che il loro prodotto sia trascurabile rispetto a E_{in} e E_{alg}.

Traiamo la conclusione che all'errore che si commette nell'effettuare una operazione in aritmetica dei calcolatori contribuisce sia l'errore inerente che l'errore algoritmico e l'errore totale è dato dalla somma di entrambi questi errori, come riportato nello schema che segue

Per poter risolvere in maniera corretta un problema al calcolatore abbiamo bisogno che il problema sia ben condizionato e che l'algoritmo utilizzato sia stabile.

In alcuni casi ad un problema mal condizionato potremo sostituire un problema equivalente, che abbia quindi la stessa soluzione, ma che sia meglio condizionato. Per gli algoritmi dovremo studiarne la stabilità e scegliere gli algoritmi più stabili.

3.9 Cenni sulla stabilità: forward and backward stablity

Due algoritmi distinti ma equivalenti dal punto di vista matematico (che quindi in aritmetica reale produrrebbero lo stesso risultato) quando eseguiti in aritmetica dei calcolatori generalmente producono risultati differenti, introducendo degli errori. E' necessario pertanto studiare ciascuno algoritmo per verificare se sia *stabile*, e quindi introduce errori piccoli, oppure *instabile*, che quindi introduce grossi errori.

Dal punto di vista più formale consideriamo un algoritmo $\mathcal{A}$ che trasforma un dato di input $x \in \mathbb{F}$ in un dato di output $y \in \mathbb{R}$

$$\mathcal{A} : x \in \mathbb{F} \longrightarrow y \in \mathbb{R},$$

dove, al solito, $\mathbb{F} = \mathbb{F}(\beta, t, m, M)$ è l'aritmetica del calcolatore. L'algoritmo $\mathcal{A}$ è costituito da una serie di passi (operazioni) ciascuno dei quali introduce un errore dovuto all'aritmetica del calcolatore. Il risultato finale non sarà pertanto l'output atteso y ma un valore $\hat{y} \in \mathbb{F}$, ovviamente perturbato (quindi affetto da errore) rispetto a y. Indichiamo pertanto con $\hat{\mathcal{A}}$ il corrispondente algoritmo eseguito in aritmetica del calcolatore

$$\hat{\mathcal{A}} : x \in \mathbb{F} \longrightarrow \hat{y} \in \mathbb{F}$$

e l'errore che si ottiene $|y - \hat{y}|/|y|$ viene detto *forward error*. Un algoritmo si dice allora *forward stable* se produce un *forward error* piccolo, ossia

$$\frac{|y - \hat{y}|}{|y|} \leq Cu$$

con u la quantità legata alla precisione del calcolatore e C una costante che assumeremo non troppo grande. In sostanza affinché l'algoritmo si possa definire *forward stable* è necessario che il *forward error* sia dello stesso ordine di grandezza di u, ossia $|y - \hat{y}|/|y| \approx u$.

Non è però possibile conoscere il *forward error* in quanto il risultato esatto y non è in generale noto (a parte per semplici problemi test). Ed anche uno studio analitico per fornirne una

stima appare troppo complesso in quanto andrebbe studiata la propagazione degli errori nella moltitudine di operazioni di cui si compone l'algoritmo. Operazioni eseguite in aritmetica del calcolatore per la quale non valendo le consuete proprietà non siamo in grado di analizzare con esattezza gli effetti.

Si deve pertanto ricorrere ad una tecnica indiretta che consenta di studiare l'algoritmo $\hat{\mathcal{A}}$ in aritmetica dei calcolatori attraverso un corrispondente algoritmo $\mathcal{A}$ in aritmetica reale (della quale conosciamo tutte le proprietà). Questa tecnica va sotto il nome di *analisi all'indietro* (o *backward analysis*), e considera le operazioni di macchina come svolte in aritmetica reale ma su dati perturbati.

In pratica, se applichiamo l'algoritmo $\mathcal{A}$ ad una perturbazione $x + \delta_x$ di x abbiamo

$$\mathcal{A} : x + \delta_x \longrightarrow y + \delta_y,$$

e tra tutte le possibili perturbazioni δ_x possiamo considerare quella che ci restituisca proprio $\hat{y} = y + \delta_y$, ossia il risultato dell'applicazione a x dell'algoritmo in aritmetica del calcolatore $\hat{\mathcal{A}}$.

Simuliamo quindi $\hat{\mathcal{A}}$ sul dato esatto x come applicazione di $\mathcal{A}$ sul dato perturbato $\hat{x} = x + \delta_x$ e studiamo l'algoritmo $\mathcal{A}$ in aritmetica reale che siamo in grado di manipolare con maggiore competenza. Verifichiamo quindi la dipendenza tra le perturbazioni sui dati in ingresso e le perturbazioni sui dati in uscita.

Il *backward error* sarà pertanto $|x - \hat{x}|/|x| = |\delta_x|/|x|$ e un algoritmo si dirà *backward stable* se per ogni dato x produce un valore $\hat{y}$ a cui corrisponde un $\hat{y} = \hat{\mathcal{A}}(x + \delta_x)$ con δ_x piccolo. In pratica, per la backward stability richiediamo che $|x - \hat{x}|/|x| \approx u$.

In generale vale una regola empirica per cui

$$\textit{errore forward} \;\lesssim\; \textit{N.ro di condizione} \cdot \textit{errore backward}$$

e pertanto possiamo affermare che la *backward stability* implica la *forward stablity* ma non il viceversa, ossia

$$\textit{backward stability} \Rightarrow \textit{forward stability} \qquad \text{ma} \qquad \textit{forward stability} \not\Rightarrow \textit{backward stability}.$$

Per *n.ro di condizione* intendiamo un indice che esprime il condizionamento del problema da risolvere e che andremo a studiare di volta in volta.

Esercizio 3.1. Si dica quali dei seguenti numeri sono in forma normalizzata e nel caso non lo siano si provveda a trasformarli in forma normalizzata:

- $x_1 = 0{,}00345$
- $x_2 = 1{,}2343 \times 10^{-3}$
- $x_3 = 0{,}23435 \times 10^{2}$
- $x_4 = 0{,}012 \times 10^{4}$
- $x_5 = 1/4$

Esercizio 3.2. Se indichiamo con $\mathbb{F}(\beta, t, m, M)$ l'insieme dei numeri di macchina, si descriva il significato dei simboli β, t, m e M.

Esercizio 3.3. Si stabilisca se i seguenti numeri appartengono all'insieme dei numeri di macchina definito da $\mathbb{F}(10, 4, -5, 5)$:

- $x_1 = 0{,}00345$
- $x_2 = 1{,}2343 \times 10^{-3}$
- $x_3 = 0{,}2343 \times 10^{2}$
- $x_4 = 0{,}00012 \times 10^{-2}$
- $x_5 = 1/4$
- $x_6 = 110{,}3 \times 10^{3}$

Esercizio 3.4. Si scriva un codice che calcoli l'*epsilon machine* del proprio calcolatore e quindi sia la *precisione* che il *numero di cifre significative.*

Esercizio 3.5. Si calcoli il *RealMin*, *RealMax*, *Epsilon machine* e numero di cifre significative per ciascuna delle 3 precisioni previste dalla standard IEEE-754 mostrandone il procedimento.

Esercizio 3.6. Dato un insieme di numeri macchina $\mathbb{F}(10, 4, -5, 5)$ se ne calcoli il *RealMin*, *RealMax* e l'*Epsilon machine*.

Esercizio 3.7. Si considerino due numeri $x_1 = 0{,}5238314$ e $x_2 = 0{,}5235739$ la cui differenza è facile calcolare essere $x_1 - x_2 = 0{,}0002575$. Si verifichi mediante un programma al calcolatore se la differenza $x_1 \ominus x_2$ effettivamente calcolata coincide con $x_1 - x_2$ (visualizzare un numero consistente di cifre decimali)

4 Alcuni problemi in aritmetica dei calcolatori

In questa sezione vediamo come l'aritmetica dei calcolatori influscie nelle modalità con le quali risolviamo alcuni problemi e sulla accuratezza delle soluzioni.

4.1 Algoritmo di Ruffini-Horner per il calcolo di un polinomio

Dal momento che l'aritmetica del calcolatore potenzialmente introduce un errore ad ogni operazione, laddove possibile è consigliabile fare in modo che gli algoritmi impieghino il minor numero possibile per svolgere un certo calcolo. Uno stesso problema infatti può essere risolto mediante algoritmi differenti, ed uno dei criteri (non l'unico) per la scelta dell'algoritmo è anche quello del minor numero di operazioni.

Inoltre il contenere il numero di operazioni contribuisce a ridurre il tempo di esecuzione dell'algoritmo.

Un esempio di riformulazione di un algoritmo per ridurre il numero di operazioni è quello del calcolo di un polinomio di grado n

$$p_n(x) = c_n + c_{n-1}x + c_{n-2}x^2 + c_{n-3}x^3 + \cdots + c_1 x^{n-1} + c_0 x^n = \sum_{j=0}^{n} c_{n-j}x^j.$$

L'esecuzione di un algoritmo secondo la formula precedente comporta $j-1$ moltiplicazioni per ciascuna potenza x^j, quindi n moltiplicazioni del tipo $c_{n-j}x^j$ e quindi n addizioni. Complessivamente il numero di moltiplicazioni è dato da

$$\sum_{j=0}^{n}(j-1) + n = \sum_{j=0}^{n} j - \sum_{j=0}^{n} 1 + n = \frac{n(n+1)}{2} - (n+1) + n = \frac{n^2}{2} + \frac{n}{2} - 1 \approx \frac{n^2}{2}.$$

Una diversa organizzazione del calcolo può però portarci a compiere un numero minore di operazioni. Per illustrare l'algoritmo di Ruffini-Horner partiamo da un polinomio di grado $n = 5$ e poniamo ogni volta in evidenza il termine x in comume

$$\begin{aligned} p_5(x) &= c_5 + c_4x + c_3x^2 + c_2x^3 + c_1x^4 + c_0x^5 \\ &= c_5 + x\big(c_4 + c_3x + c_2x^2 + c_1x^3 + c_0x^4\big) \\ &= c_5 + x\big(c_4 + x\big(c_3 + c_2x + c_1x^2 + c_0x^3\big)\big) \\ &= c_5 + x\big(c_4 + x\big(c_3 + x\big(c_2 + c_1x + c_0x^2\big)\big)\big) \\ &= c_5 + x\big(c_4 + x\big(c_3 + x\big(c_2 + x\big(c_1 + c_0x\big)\big)\big)\big) \end{aligned}$$

ed a questo punto possiamo partire dalle parentesi più interne mediante un algoritmo ricorsivo

$$
\begin{aligned}
p &\leftarrow c_0 \\
p &\leftarrow c_1 + p * x \\
p &\leftarrow c_2 + p * x \\
p &\leftarrow c_3 + p * x \\
p &\leftarrow c_4 + p * x \\
p &\leftarrow c_5 + p * x
\end{aligned}
$$

Il procedimento può essere applicato ad un generico prolinomio di grado n mediante un semplice codice Python del tipo

```
p = c[0]
for j in range(1,n+1):
    p = c[j] + x*p
```

che comporta l'effettuazione di sole n moltiplicazioni ed n addizioni.

4.2 L'approssimazione della derivata

Nelle sezioni precedenti ci siamo occupati degli errori introdotti dalla particolare rappresentazione dei numeri al calcolatori. La risultanza di tali errori (inerente ed algoritmico) è detta anche *errore di arrotondamento* in quanto dipende da come il calcolatore arrotonda (towards zero o towards nearest) i numero non di macchina.

Un errore non trascurabile dipende anche da come risolviamo il problema in quanto spesso il metodo utilizzato approssima la soluzione ma non la calcola in maniera esatta. L'errore dovuto al metodo prende il nome di *errore analitico*.

I due errori (*di arrotondamento* ed *analitico*) agiscono contemporaneamente sul risultato, spesso in maniera differente. Non si può fare un'analisi generale, ma attraverso un esempio possiamo effettuare delle osservazioni significative.

Come è noto la derivata di una funzione $f(x)$ è data da

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

ma tale definizione non può essere utilizzata per calcolare numericamente il valore di f' in un punto x. E' però possibile approssimare $f'(x)$ scegliendo un opportuno $h > 0$ piccolo e calcolando quella che viene chiamata la *differenza finita in avanti*

$$f'_h(x) = \frac{f(x+h) - f(x)}{h}.$$

Per vedere quanto bene $f'_h(x)$ approssima $f'(x)$ osserviamo, attraverso lo sviluppo in serie di Taylor che

$$f(x+h) = f(x) + hf'(x) + h^2 \frac{1}{2} f''(c_x), \quad c_x \in (x, x+h)$$

per cui

$$f'(x) - \frac{f(x+h) - f(x)}{h} = -h \frac{1}{2} f''(c_x)$$

e quindi

$$\left|f'(x) - f'_h(x)\right| \leq \frac{h}{2}M, \quad M = \max_{c\in(x,x+h)}\left|f''(c)\right|.$$

Se ne deduce pertanto che l'errore commesso approssimando $f'(x)$ con la differenza in avanti $f'_h(x)$ è proporzionale ad h e sarà pertanto sufficiente utilizzare un h sufficientemente piccolo per ottenere un errore piccolo quanto si vuole. Tale analisi non tiene però conto degli errori, che chiamiamo di arrotondamento, che necessariamente si introducono quando lavoriamo al calcolatore.

Per studiare gli effetti degli errori di arrotondamento assumiamo che i valori effettivamente calcolati $\tilde{f}(x)$ e $\tilde{f}(x+h)$ differiscano dai valori esatti $f(x)$ e $f(x+h)$ per delle piccole quantità δ_1 e δ_2, ossia che

$$\tilde{f}(x) = f(x) + \delta_1, \quad \tilde{f}(x+h) = f(x+h) + \delta_2, \quad |\delta_1|, |\delta_2| \leq \varepsilon$$

con ε una quantità piccola che possiamo (per semplicità) assumere dell'ordine di grandezza della precisione di macchina. La differenza finita effettivamente calcolata sarà data pertanto da

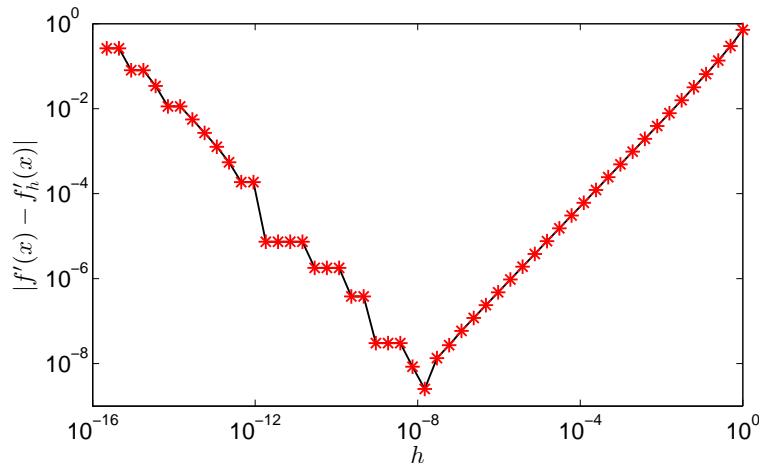
$$\tilde{f}'_h(x) = \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h} = \frac{f(x+h) + \delta_2 - f(x) - \delta_1}{h} = f'_h(x) + \frac{\delta_2 - \delta_1}{h}$$

e l'errore di arrotondamento, ossia la distanza tra la differenza finita teorica ed il suo valore effettivamente calcolato, è dato da

$$\left|f'_h(x) - \tilde{f}'_h(x)\right| = \frac{\left|\delta_2 - \delta_1\right|}{h} \leq \frac{\left|\delta_2\right| + \left|\delta_1\right|}{h} \leq \frac{2\varepsilon}{h}.$$

L'errore che complessivamente si commette al calcolatore è pertanto dato da

$$\begin{aligned}\left|f'(x) - \tilde{f}'_h(x)\right| &= \left|f'(x) - f'_h(x) + f'_h(x) - \tilde{f}'_h(x)\right| \\ &\leq \underbrace{\left|f'(x) - f'_h(x)\right|}_{\text{Err. analitico}} + \underbrace{\left|f'_h(x) - \tilde{f}'_h(x)\right|}_{\text{Err. arrotond.}} \leq \frac{h}{2}M + 2\frac{\varepsilon}{h}.\end{aligned}$$

Come si può notare, al diminuire di h il primo termine decresce mentre il secondo cresce. Esiste quindi un valore soglia $\bar{h}$ oltre il quale non risulta conveniente diminuire ulteriormente il passo h poiché l'errore di arrotondamento tende a predominare sull'errore analitico, come si evince dalla figura seguente dove abbiamo rappresentato l'errore nel calcolo in $x = 1$ della derivata della funzione esponenziale $f(x) = \mathrm{e}^x$.

Tale valore di soglia può essere agevolmente stimato andando ad imporre che le maggiorazioni dei due errori (analitico e di arrotondamento) siano uguali (o comunque dello stesso ordine di grandezza)

$$\frac{h}{2}M = 2\frac{\varepsilon}{h} \quad \Longrightarrow \quad h^2 = 4\frac{\varepsilon}{M} \quad \Longrightarrow \quad h = \sqrt{\varepsilon}\frac{2}{\sqrt{M}}$$

In mancanza di informazioni più precise sul valore di M e sugli errori effettivamente commessi nel calcolo di $f(x)$ e $f(x+h)$ si può assumere $\bar{h} = \sqrt{\varepsilon}$ come valore soglia indicativo per il passo h. Dal momento che nell'aritmetica in doppia precisione $\varepsilon \approx 2.2 \times 10^{-16}$, il valore soglia per la differenza finita in avanti è dato da $\bar{h} \approx 1.4 \times 10^{-8} \approx 2^{-26}$.

Riferimenti bibliografici

[1] Goldberg David. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[2] Overton Michael L. Numerical Computing with IEEE Floating Point Arithmetic. SIAM, 2001.